Heritage

Concept de l'héritage

Une classe peut hériter d'une autre classe. Cela permet de créer de nouvelles classes mais avec une base existante.

- Une classe qui hérite d'une autre classe est appelée sous-classe, classe enfant, classe fille ou encore classe dérivée.
- Une classe dont hérite(nt) une ou plusieurs classes est appelée super-classe ou classe mère.

L'objectif de cette hiérarchie logique est de permettre à la classe enfant de récupérer toutes les méthodes et les attributs de la classe mère.

L'héritage en Python

Pour définir un héritage, il est nécessaire de le définir sur la classe enfant.

```
class ClasseMere:
    pass

class ClasseEnfant(ClasseMere):
    pass
```

La méthode « issubclass » permet de tester la hiérarchie d'héritage entre les types.

```
test = issubclass(ClasseEnfant, ClasseMere)
```

La méthode « super() »

La méthode « super() » permet d'utiliser les propriétés et les méthodes de la classe mère au sein de la classe enfant.

Elle permet en autre de faire référence à la méthode « __init__ » de la classe mère au sein de l'enfant.

```
class ClasseMere:

    def __init__(self, msg):
        self.__msg = msg


class ClasseEnfant(ClasseMere):

    def __init__(self, msg, val):
        super().__init__(msg)
        self.__val = val


o1 = ClasseMere("Hello")
o2 = ClasseEnfant("Hi", 42)
```

Redéfinition de méthodes

Les méthodes de la classe mère peuvent être redéfinies par les enfants de la classe.

Cela permet de définir un comportement spécialisé pour une classe enfant.

```
class ClasseMere:

    def ma_methode(self):
        print("Méthode de la classe mère")


class ClasseEnfant(ClasseMere):

    def ma_methode(self):
        print("Méthode redéfinie dans la classe fille")
```

Exemple de mise en place (1/2)

La classe mère

```
class Animal:

    def __init__(self, nom):
        self.__nom = nom

    @property
    def nom(self):
        return self.__nom

    @nom.setter
    def nom(self, value):
        self.__nom = value

    def dormir(self):
        print(self.nom, "dort!")
```

La classe enfant

```
class ClasseStatique:

    x = 1337


print(ClasseStatique.x)      # 1337

ClasseStatique.x = 42
print(ClasseStatique.x)      # 42

c = ClasseStatique()
print(c.x)                   # 42
print(c.__dict__)            # {}

c.x = 13
print(c.x)                   # 13
print(c.__dict__)            # {'x': 13}

print(ClasseStatique.x)      # 42
```

Exemple de mise en place (2/2)

```
# Création d'un object "Chat"
c = Chat("Le chat", "Noir")

# Utilisation des propriétés
msg = c.nom + " est de couleur " + c.couleur
print(msg)

# Utilisation des méthodes
c.ronronner()
c.dormir()
```

Le polymorphisme

Le concept du polymorphisme signifie qu'un objet peut avoir plusieurs types.

La méthode « isinstance » permet de tester les types d'un objet.

```
# Création d'un object "Chat"
c = Chat("Le chat", "Noir")

# Possede le type "Chat"
test1 = isinstance(c, Chat)          # True

# Possede le type "Animal"
test2 = isinstance(c, Animal)        # True

# Ne possede pas le type "Voiture"
test3 = isinstance(c, Voiture)        # False
```